WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Saturday, March 20, 2004

Hide?	Set Name	Query	Hit Count
	DB=USPT; PLUR=YES; OP=ADJ		
☐	L22	L20 and ((toleranc$ or threshold$ or percentag$ or weight$ or metric) near12 (server near4 (load$ or balanc$ or distribut$)))	4
☐	L21	L20 and ((toleranc$ or threshold$ or percentag$) near12 (server near4 (load$ or balanc$ or distribut$)))	3
☐	L20	l19 or l7	14
☐	L19	l17 and (l3 same l4)	13
☐	L18	L17 same l3 same l4	0
☐	L17	server near2 request$ near2 (number$ or total$)	217
☐	L16	(6665702 or 6249801)[pn]	2
☐	L15	(6665763 or 6249801)[pn]	2
	DB=EPAB,DWPI; PLUR=YES; OP=ADJ		
☐	L14	L13 and ((server near4 request$) near8 (distribut$ or balanc$))	3
☐	L13	(server near2 (capabilit$ or capacit$ or characteri$)) same (server near4 (select$ or assign$))	31
	DB=USPT; PLUR=YES; OP=ADJ		
☐	L12	L11 and ((server near4 request$) near8 (distribut$ or balanc$))	43
☐	L11	(server near2 (capabilit$ or capacit$ or characteri$)) same (server near4 (select$ or assign$))	182
☐	L10	6205477[uref]	4
☐	L9	l7 and l8	7
☐	L8	(709/226 or 709/203 or 709/220 or 709/223 or 718/105 or 718/104 or 718/102).ccls.	5428
☐	L7	l5 same l1	8
☐	L6	L5 and l1	43
☐	L5	l3 same L4	182
☐	L4	server near4 (select$ or assign$)	7024
☐	L3	server near2 (capabilit$ or capacit$ or characteri$)	2240
☐	L2	6205477[pn]	1
☐	L1	(server near4 request$) near8 (distribut$ or balanc$)	753

END OF SEARCH HISTORY

US006351775B1

(12) United States Patent
Yu

(10) Patent No.: US 6,351,775 B1
(45) Date of Patent: Feb. 26, 2002

(54) LOADING BALANCING ACROSS SERVERS IN A COMPUTER NETWORK

(75) Inventor: **Philip Shi-Lung Yu**, Chappaqua, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/866,461**

(22) Filed: **May 30, 1997**

(51) **Int. Cl.**[7] **G06F 15/173**

(52) **U.S. Cl.** **709/238**; 709/239; 709/240; 709/241; 709/242; 370/237; 370/400

(58) **Field of Search** 709/242, 239, 709/240, 241, 238; 370/237, 400

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,371,852	A	12/1994	Attanasio et al.	
5,517,620	A *	5/1996	Hashimoto et al.	709/242
5,526,414	A *	6/1996	Bedard et al.	379/221
5,544,313	A *	8/1996	Shachnai et al.	709/219
5,828,847	A *	10/1998	Gehr et al.	709/239
5,864,535	A *	1/1999	Basilico	370/231
5,930,348	A *	7/1999	Regnier et al.	379/221
6,078,943	A *	6/2000	Yu	709/105
6,091,720	A *	7/2000	Bedard et al.	370/351

FOREIGN PATENT DOCUMENTS

JP 8-214063 8/1996

OTHER PUBLICATIONS

M. Colajanni et al., "Scheduling Algorithms for Distributed Web Servers", RC 20680 (91683) Jan. 6, 1997, Computer Science/Mathematics, Research Report, 29 pages.

T. Brisco, "DNS Support for Load Balancing", Apr. 1995, 6 pages, Network Working Group, Rutgers University.

Daniel M. Dias et al., "A Scalable and Highly Available Web Server", (not dated), 8 pages, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, N. Y. 10598.

Eri D. Katz et al., "A scalable HTTP server: The NCSA prototype", 1994, pp. 155–164, vol. 27, Computer Networks and ISDN Systems.

* cited by examiner

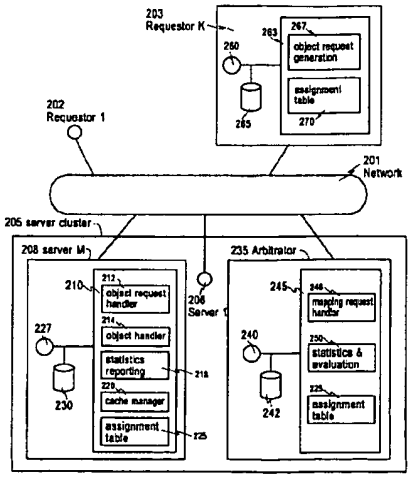
Primary Examiner—Krisna Lim

(74) *Attorney, Agent, or Firm*—F. Chau & Associates, LLP

(57) **ABSTRACT**

A dynamic routing of object requests among a collection or cluster of servers factors the caching efficiency of the servers and the load balance or just the load balance. The routing information on server location can be dynamically updated by piggybacking meta information with the request response. To improve the cache hit at the server, the server selection factors the identifier (e.g. URL) of the object requested. A partitioning method can map object identifiers into classes; and requester nodes maintain a server assignment table to map each class into a server selection. The class-to-server assignment table can change dynamically as the workload varies and also factors the server capacity. The requester node need only be informed on an "on-demand" basis on the dynamic change of the class-to-server assignment (and thus reduce communication traffic). In the Internet, the collection of servers can be either a proxy or Web server cluster and can include a DNS and/or TCP-router. The PICS protocol can be used by the server to provide the meta information on the "new" class-to-server mapping when a request is directed to a server based on an invalid or obsolete class-to-server mapping. DNS based routing for load balancing of a server cluster can also benefit. By piggybacking meta data with the returned object to reassign the requester to another server for future requests, adverse effects of the TTL on the load balance are overcome without increasing traffic.

75 Claims, 15 Drawing Sheets



First Hit Fwd Refs

Generate Collection Print

L22: Entry 2 of 4 File: USPT Feb 26, 2002

DOCUMENT-IDENTIFIER: US 6351775 B1
TITLE: Loading balancing across servers in a computer network

Abstract Text (1):
A dynamic routing of object requests among a collection or cluster of servers factors the caching efficiency of the servers and the load balance or just the load balance. The routing information on server location can be dynamically updated by piggybacking meta information with the request response. To improve the cache hit at the server, the server selection factors the identifier (e.g. URL) of the object requested. A partitioning method can map object identifiers into classes; and requester nodes maintain a server assignment table to map each class into a server selection. The class-to-server assignment table can change dynamically as the workload varies and also factors the server capacity. The requester node need only be informed on an "on-demand" basis on the dynamic change of the class-to-server assignment (and thus reduce communication traffic). In the Internet, the collection of servers can be either a proxy or Web server cluster and can include a DNS and/or TCP-router. The PICS protocol can be used by the server to provide the meta information on the "new" class-to-server mapping when a request is directed to a server based on an invalid or obsolete class-to-server mapping. DNS based routing for load balancing of a server cluster can also benefit. By piggybacking meta data with the returned object to reassign the requester to another server for future requests, adverse effects of the TTL on the load balance are overcome without increasing traffic.

Detailed Description Text (19):
FIG. 10 depicts an example of the Statistics and Evaluation routine (250). As depicted, in step 1010, statistics collection requests are communicated to all servers to get the CS(j,i), for i=1, . . . , N, from server j, for j=1, . . . , M. In step 1020, CA(i) is calculated for each class (the total number of requests across all servers for each class i). In step 1030, SA(j) is calculated for each server j, (the total number of requests to the assigned classes of each server j). In step 1040, the upper threshold, TH, of the load on a server is calculated. TH is preferably defined to be a fraction (d) above the average load. For example, d can be 0.2, which means, the target for load balancing is to have none of the servers exceeding 20% of the average. In step 1050, if any server's load exceeds the threshold TH, the Reassignment Routine is invoked to adjust the class-to-server assignment so that better load balancing can be achieved. A detailed example of the Reassignment Routine will be described with reference to FIG. 11. In step 1070, the statistics collection timer is reset to the length of the desired statistics collection interval.

Detailed Description Text (20):
FIG. 11 depicts an example of the Reassignment Routine (step 1060). In step 1110, TO includes the set of servers that exceed the loading threshold (TH). In step 1115, let k be the index of the most loaded server in TO. In step 1120, TU includes the set of servers that have not exceeded the loading threshold. In step 1125, let l be index of the least loaded server in TU; and in step 1130, let i be the class assigned to server k with the smallest class load, CA(i). In step 1135, if reassigning class i to sever l does not cause the load of server l to exceed the threshold, i.e., SA(l)+CA (i)<=TH, class i is reassigned to server l from server k

in step 1140 (by changing C(i) to l) and SA(l) and SA(k); and in step 1145, SA(l) and SA(k) are updated to reflect the class reassignment. Specifically, SA(l)is incremented by CA(i) and SA(k) is decremented by CA(i). Otherwise, in step 1160, server l is deleted from TU as it is no longer able to accept load from the overloaded servers. In step 1150, if the load of server k still exceeds the threshold, i.e., SA(k)>TH, step 1130 is re-executed. Otherwise, in step 1155, server k is deleted from TO, since its load no longer exceeds the threshold. In step 1170, if TO is not empty, step 1115 is re-executed. In step 1165, if TU is not empty, step 1125 is re-executed.

Detailed Description Text (21):
Those skilled in the art will readily appreciate that various alternative embodiments and extensions to the present invention can be used within the spirit and scope thereof. For example, in step 1140, the reassignment is a simple greedy approach to allow the movement of a single Class (FIG. 3) from server k to server l to reduce a load imbalance. An extension would be to allow for a swap or exchange of one Class from server k with another class from server l if it can improve the load balance. In step 1135, the reassignment occurs only if server l does not exceed the load threshold. The criterion can be relaxed to instead measure whether the total overload is reduced. Furthermore, if any of the class load, CA(i), exceeds TH, it can be assigned to multiple servers where each of these servers will get a fraction of the requests for that class. The arbitrator can assign servers to requesters for that class probabilistically according to the fraction assigned to the server. A similar reassignment can be implemented at the server (208).

Detailed Description Text (30):
In a preferred embodiment, the DNS (167) collects the number of requests issued from each requester and will generate a requester-to-server assignment table to balance the load among the servers. (For heterogeneous servers, the assigned load can be made proportional to the server's processing capacity). When a (name-to-address) mapping request arrives at the DNS (167), a server (161 . . . 163) is assigned based on the requester name (or IP address) in the assignment table. The mapping is hierarchical and multi-level, e.g., URL=>Class=>virtual server=>server. The DNS (167) can collect the load statistics and update the assignment table (225) based on a measurement interval (much) smaller than the TTL. Thus, a new assignment table can be quickly generated, to better reflect load conditions. All servers (161 . . . 163) get the up-to-date version of the assignment table (225) from the DNS (167). As before, the requesters (110 . . . 153) need not be informed of the change; they can still send requests based on the previous (name-to-address) mapping. However, if a server receives a request from a requester that is no longer assigned to that server, the server will inform the requester of the server (161 . . . 163) to which future requests should be issued. The current request will still be served and the new assignment information can be piggybacked, e.g., using PICS or a similar mechanism, with the response or returned object. When a server is overloaded, it can send an alarm signal to the DNS (167). Each time an alarm is received, the DNS (167) can recalculate the assignment table to reduce the number of requesters assigned to any overloaded servers. The requesters can also be partitioned into classes so that the assignment table can then become a class-to-server assignment.

US006601084B1

(12) **United States Patent**
Bhaskaran et al.

(10) **Patent No.: US 6,601,084 B1**
(45) **Date of Patent: Jul. 29, 2003**

(54) **DYNAMIC LOAD BALANCER FOR MULTIPLE NETWORK SERVERS**

(75) Inventors: **Sajit Bhaskaran**, Sunnyvale, CA (US); **Abraham R. Matthews**, San Jose, CA (US)

(73) Assignee: **Avaya Technology Corp.**, Basking Ridge, NJ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/992,038**

(22) Filed: **Dec. 19, 1997**

(51) **Int. Cl.**[7] **G06F 9/00**
(52) **U.S. Cl.** **709/105**; 709/223
(58) **Field of Search** 709/105, 223–226, 709/229

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,283,897 A	*	2/1994	Georgiadis et al.	709/105
5,473,599 A		12/1995	Li et al.	370/219
5,586,121 A		12/1996	Moura et al.	370/404
5,612,865 A	*	3/1997	Dasgupta	700/79
5,612,897 A		3/1997	Rege	709/219
5,634,125 A	*	5/1997	Li	707/203
5,687,369 A	*	11/1997	Li	707/203
5,754,752 A		5/1998	Sheh et al.	714/4
5,796,941 A	*	8/1998	Lita	713/201
5,812,819 A		9/1998	Rodwin et al.	703/23
5,815,668 A		9/1998	Hashimoto	709/238
5,835,696 A		11/1998	Hess	714/10
5,936,936 A		8/1999	Alexander, Jr. et al.	370/216
5,951,634 A	*	9/1999	Sitbon et al.	709/105

FOREIGN PATENT DOCUMENTS

WO	WO 99/32956	7/1999	G06F/0/00

OTHER PUBLICATIONS

Internet. "Quasi–Dynamic Load–Balancing (QDLB) Methods." Apr. 25, 1995, pp. 2 and 5.
(INTERNET) "Dynamic Load–Balancing Methods", Apr. 25, 1995.*
(Becker) Becker, Wolfgang. "Dynamic Load Balancing for Parallel Database Processing", 1997.*
(Omiecinski) Omiencinski, Edward. "Performance Analysis of a Load Balancing Hash–Join Algorithm for a Shared Memory Multiprocessor", 1991.*
(IBM) "Value–Oriented Approach fto Selecting Buckets for Data Redistribution", May 1, 1993.*

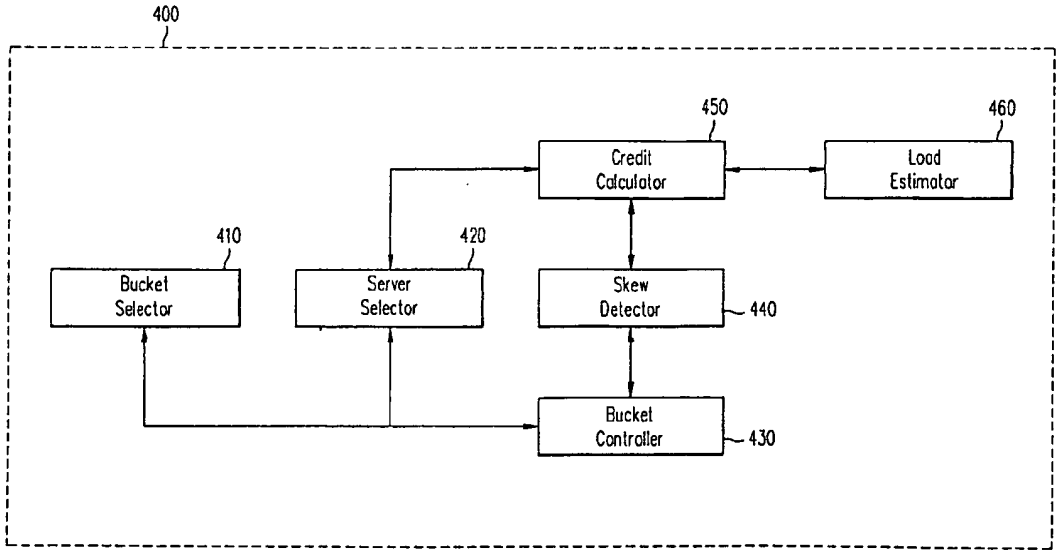
* cited by examiner

Primary Examiner—Alvin Oberley
Assistant Examiner—Lewis A. Bullock, Jr.
(74) *Attorney, Agent, or Firm*—David Volejnicek

(57) **ABSTRACT**

The present invention provides methods and systems for balancing the load on a plurality of servers using a load balancing algorithm which continuously examines the loads on the plurality of servers and makes adjustments in the loads accordingly. Among the factors considered in the load balancing are the power of each server relative to other servers, the load on each server relative to the other servers, and a "credit" for each server based on their power and load.

37 Claims, 10 Drawing Sheets



First Hit Fwd Refs

Generate Collection Print

L22: Entry 1 of 4 File: USPT Jul 29, 2003

DOCUMENT-IDENTIFIER: US 6601084 B1
TITLE: Dynamic load balancer for multiple network servers

Brief Summary Text (17):
The server fault tolerance mechanism in the load balancer detects when a server goes down and redistributes the load to the new set of operational servers. Additionally, when previously non-operational server becomes operational, the traffic is redistributed over the new set of operational servers. This redistribution is done in such a manner as not to disrupt any existing client-server connections.

Detailed Description Text (24):
Specifically, the credit calculator 450 at periodic intervals calculates a weighted load for each server, as shown in equation 2:

Detailed Description Text (25):
where Li is the load on server i (numbered from 0 to S-1) and Li is the weighted load on server i.

Detailed Description Text (27):
Specifically, the credit calculator 450 calculates the credit of each server, Ki, by subtracting the normalized weighted load of the server from the weight of the server, as shown in equation 4:

Detailed Description Text (36):
At time epoch t, load estimator 460 gets the load vector and converts it into a scalar quantity L.sub.i. Since the servers are of different computing capacity, load estimator 460 normalizes the loads before they can be compared. Intuitively, a load of 100 units to a server of capacity 10 should equal a load of 200 units to a server of capacity 20. Thus, load estimator 460 normalizes based on the inverse of the server weights. Server credits are computed periodically to decide which server has the maximum capacity available. In Table 1, rows 3 to 7, show the computation of server credits by credit calculator 450 at epoch t when the raw loads are all equal. Row 3: Current load per server (L.sub.i). This is computed from a load vector. Row 4: The weighted load for each server is computed in accordance with equation 2. Note that even though each server has the same load, it represents a larger load for server 660 than for server 670 or server 680. The effective load for an idealized server is the sum of the normalized loads for all servers; Row 5: To relate the load to capacity, the load normalizer LN is computed in accordance with equation 3. Evidently, this value has to be a function of the current load. Row 6: A normalized estimate of computational resources used by the current load is subtracted from the server's weight to arrive at the server's credits (K.sub.i) in accordance with equation 4. Note that the idealized server's credits can be computed in two different ways. In one method, we use the same formula that we use for each server. Alternately, we could sum up the credits of all servers. Both methods should come out to be equal (or very nearly equal, due to integer arithmetic). Row 7: Finally, an estimate of computational use by a new flow is needed for each server. Towards this goal, we compute the flow weight in accordance with equation 5. This is also the amount that we must deduct from the idealized server's credits when a flow is assigned to it. Whenever a flow is assigned to a

server at a later time epoch (say t+k), the computational resource to be subtracted from the server's credits is shown in Table 1, rows 8 (in accordance with equation 6). This credit adjustment is performed by server selector 420 whenever a bucket-to-server assignment occurs. Row 8: The computational resource that a flow consumes for each server is computed in accordance with equation 6. This is the value to deduct from the server's credits when a new bucket is assigned to the server. Note that we can assign three flows to server 680, two flows to server 670, and, one flow to server 660, and each individual server's credits would be diminished by the same amount.

Detailed Description Text (39):
As explained above, load estimator 460 provides credit calculator 450 with an estimate of the load on each server. Common measures of the load on a server are based on the number of data requests, the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, the number of bytes sent by the server, the processor activity on the server, and the memory utilization on the server. Because load balancer 400 is primarily designed to balance the network load on a plurality of servers, load estimator 460 typically uses load measures based on network traffic to the server. Thus, most embodiments of load estimator 460 use load measures based on the number of data requests received by the server, the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, or the number of bytes sent by the server.

Detailed Description Text (40):
One embodiment of the load estimator 460 uses a combination of the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, and the number of bytes sent by the server. This embodiment of load estimator 460 forms a load vector with four scalar quantities corresponding to the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, and the number of bytes sent by the server. This embodiment of load estimator 460 calculates the load of a server using the cross product of the load vector with a weighting vector, having four scalar weights. Thus, the load of a server is calculated as shown in equation 8: ##EQU2##

Detailed Description Text (60):
In the various embodiments of this invention, methods and structures have been described that provide dynamic load balancing and server fault tolerance for a plurality of servers. By monitoring the load on the servers and dynamically allocating buckets based on the credit of each server, embodiments of the present invention can maintain a balanced load on the servers. Furthermore, by monitoring the status of the servers, embodiments of the present invention can gracefully handle server faults by distributing the load of a down server among the remaining operational servers.

Detailed Description Text (61):
The various embodiments of the structures and methods of this invention that are described above are illustrative only of the principles of this invention and are not intended to limit the scope of the invention to the particular embodiments described. In view of this disclosure, those skilled in the art can define other bucket selectors, server selectors, skew detectors, credit calculators, load estimators, bucket controllers, server fault detectors, load vectors, weighting vectors, credit allocation rules, load estimation rules, fault detection rules, buckets, bucket states, or network configurations, and use these alternative features to create a method, circuit, or system according to the principles of this invention.

CLAIMS:

20. A method of assigning a plurality of data requests among a plurality of servers, said method comprising: determining a weight of each server as a ratio of a power rating of the server and a sum of the power ratings of all of the servers; determining a weighted load of each server as a ratio of a load of the server and the weight of the server; determining a load normalizer as a minimum one of absolute values of the weighted loads of the servers; determining a credit of each server as a difference between the weight of the server and a ratio of the weighted load of the server and the load normalizer; assigning each of said data requests to a bucket of a plurality of buckets; and assigning buckets of said plurality of buckets containing a data request to servers of said plurality of servers that have greatest said credits.

28. The method of claim 20 wherein: a load of a server comprises a plurality of ones of: a number of data requests received by the server, a number of data packets received by the server, a number of bytes received by the server, a number of data packets sent by the server, and a number of bytes sent by the server.

29. A load balancer for balancing the load due to a plurality of data requests on a plurality of servers, said load balancer comprising: a plurality of buckets; a bucket selector configured to assign each of said data requests to a bucket of said plurality of buckets; a credit calculator configured to determine a weight of each server as a ratio of a power rating of the server and a sum of the power ratings of all of the servers, to determine a weighted load of each server as a ratio of a load of the server and the weight of the server, to determine a load normalizer as minimum one of absolute values of the weighted loads of the servers, and to determine a credit of each server as a difference between the weight of the server and a ratio of the weighted load of the server and the load normalizer; and a server selector configured to assign buckets of said plurality of buckets containing a data request to servers of said plurality of servers that have greatest said credits.

33. The load balancer of claim 32 wherein: the credit calculator is configured to scale said weight of each server by a weight scale and to scale said weighted load of each server and the flow adjustment by a load scale.

35. The load balancer of claim 29 wherein: the credit calculator is configured to scale said weight of each server by a weight scale and to scale said weighted load of each server by a load scale.

37. The load balancer of claim 29 wherein: a load of a server comprises a plurality of ones of: a number of data requests received by the server, a number of data packets received by the server, a number of bytes received by the server, a number of data packets sent by the server, and a number of bytes sent by the server.

US005341477A

United States Patent [19]

Pitkin et al.

[11] Patent Number: 5,341,477

[45] Date of Patent: Aug. 23, 1994

[54] **BROKER FOR COMPUTER NETWORK SERVER SELECTION**

[75] Inventors: **Richard P. Pitkin**, Lowell; **John P. Morency**, Chelmsford, both of Mass.

[73] Assignee: **Digital Equipment Corporation**, Maynard, Mass.

[21] Appl. No.: **103,722**

[22] Filed: **Aug. 6, 1993**

Related U.S. Application Data

[63] Continuation of Ser. No. 924,390, Aug. 3, 1992, abandoned, which is a continuation of Ser. No. 314,853, Feb. 24, 1989, abandoned.

[51] **Int. Cl.**[5] **G06F 13/00**; G06F 15/16

[52] **U.S. Cl.** **395/200**; 395/325; 364/DIG. 1; 364/281.6; 364/284.4; 364/264; 364/940.61

[58] **Field of Search** 395/200, 325, 725

[56] **References Cited**

U.S. PATENT DOCUMENTS

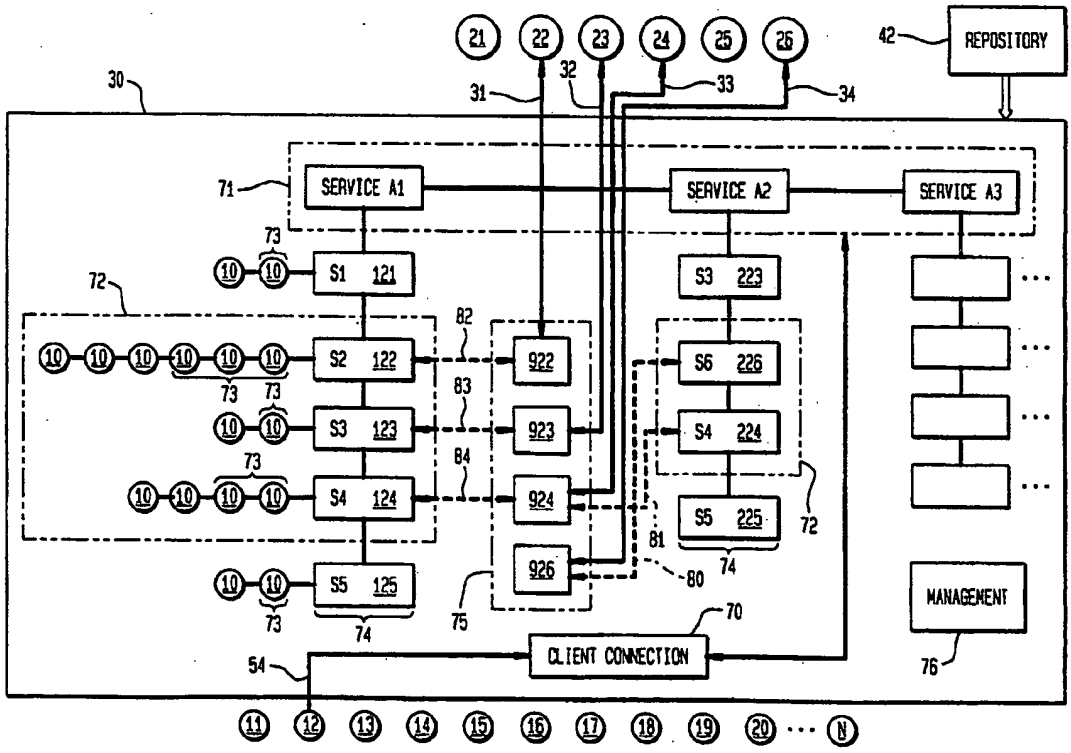
4,757,267	7/1988	Riskin	379/113
4,780,821	10/1988	Crossley	364/200
4,800,488	1/1989	Agrawal et al.	364/200
4,858,120	8/1989	Samuelson	364/401
4,897,781	1/1990	Chang et al.	364/200
4,914,571	4/1990	Baratz et al.	364/DIG. 1
4,949,248	8/1990	Caro	364/200
5,005,122	4/1991	Griffin et al.	364/200

Primary Examiner—Eddie P. Chan
Attorney, Agent, or Firm—Kenyon & Kenyon

[57] **ABSTRACT**

In a computer network, a broker mechanism allocates a plurality of servers, each having an available resource capacity, to a plurality of clients for delivering one of several services to the clients. The broker operates by monitoring a subset of all available servers capable of delivering the requested service. The allocation is based on developing a network policy for the plurality of servers by collecting a local policy for each of the servers. The broker receives client requests for the services and based on the network policy and available resource capacity suggests one of the servers, monitors in its subset for that particular service, to one of the clients making a request. The server suggested enforces its local policy by not allowing any connections exceeding its available resource capacity.

24 Claims, 10 Drawing Sheets



First Hit Fwd Refs

End of Result Set

☐ Generate Collection Print

L22: Entry 4 of 4 File: USPT Aug 23, 1994

DOCUMENT-IDENTIFIER: US 5341477 A
TITLE: Broker for computer network server selection

Brief Summary Text (20):
In this embodiment, the broker assigns servers in a round robin fashion to ensure that the loss of a single server does not have a major impact upon clients that request access at similar times. The application of the round robin server distribution, however, is regulated by use of a "scan weight" parameter for each server. The scan weight is defined as the number of client requests which can be satisfied by a server before that server is removed from the preview window.

Detailed Description Text (3):
Referring to FIG. 1, there is shown a model block diagram of a system enhanced by the present invention. In a network 5, a plurality of users or clients, generally designated 10, each have access to a plurality of servers, generally designated 20. Some number of the servers 20 can provide access to services A.sub.1 -A.sub.n requested by one of the clients 10. When the client 10 requests access to a service via one of the servers 20, a connection is made from the selected client 10 through the server 20. As can be seen in FIG. 1, a plurality of servers 20 are available to make the requested connection. Therefore, it becomes a problem to know which client 10 should connect to which server 20. Further, the resource capacity of the server 20 must be known in order to make an appropriate connection. Because the server resources have various and different capabilities, i.e., small CPU size, large CPU size, used and unused capacities, different serial line speeds, etc., different servers 20 are more appropriate for a particular client request. Also, it is necessary to know the service level desired by the client 10.

Detailed Description Text (5):
FIG. 2A conceptually illustrates the architectural diagram of FIG. 2. A broker mechanism 30 is used to suggest to clients 11-19 an appropriate server 21-26 for delivering the requested service (service A1 or A2). Further, individual servers 23, 24 and 25 can provide more than one service. While FIG. 2A only illustrates two services, it is apparent that the broker 30 can suggest servers to clients requesting any number of different services. When a particular client, shown as client 13, requests access to service A1, the client 13 places a request with the broker 30 on path 54. The broker 30 suggests an appropriate server from server set 21-24 to the client 13.

Detailed Description Text (19):
A server connection section 75 of the broker 30 establishes communication paths 31-34 with the servers referred to in the preceding paragraph. The communication paths 31-34 allow the broker to poll each coupled server (22, 23, 24, 26) to receive its status. The status of the servers 22, 23, 24 and 26 is stored in connection entries 922, 923, 924 and 926, respectively, within the server status block. In response to subsequent client requests for service, the broker examines these connection entries to determine each coupled server's capacity or availability to deliver a requested service, as will be described below with reference to FIGS. 5 and 5A. The server connection section 75 therefore supports a list of connection entries 922-

926 that map (80-84) to a select number of server entries in the server lists 74 for each of the services in service list 71. As shown in FIGS. 4, a server 24 can overlap between service offerings. The server connection section 75 may therefore have several server lists 74 with entries pointing to a single server connection. This pooling of server connections is important for providing multiple service offerings from one broker location.

Detailed Description Text (29):
Referring again to FIG. 4, because the capacity of each server to deliver a given service may not be equal depending upon the network policy, an additional parameter termed scan weight 73 is added. The scan weight 73 is the number of client requests which can be satisfied from a server before the particular server is removed from the preview window 72. Each server is assigned a particular scan weight value for each service by the network manager. The scan weight 73 allows the network manager to apply the network policy to more efficiently assign clients 10 to servers in a round-robin manner. The values chosen for the scan weight are developed as part of the capacity planning modelled in FIG. 3.

Detailed Description Text (30):
An example of scan weight operation is given in the server list 74 for service A.sub.1 as shown in FIG. 4. Five servers 21, 22, 23, 24, and 25 having server entries 121-125, respectively, are determined to have the following capacities to access a service for a client: server 21,2 clients(10); server 22,6 clients(10); server 23,2 clients(10); server 24,4 clients(10); and server 25,2 clients(10). Without scan weight, the broker mechanism 30, using a purely round robin method of assigning servers, would reduce the list of five available servers 21-25 to just two available servers 22 and 24 after only 10 client requests have been suggested. This occurs because the servers having the least capacity, i.e. 21, 23 and 25, are suggested as often as the other servers, thus quickly using their capacity. After the first ten requests, only servers 22 and 24 have any capacity remaining. Further, only one server 22 would be available after the next four client requests. This rapid reduction in the available servers can produce single failure points which may disrupt a network.

Detailed Description Text (31):
A scan weight parameter 73 is therefore determined based on an equal distribution of the server capacities to handle client requests. The scan weight 73 controls the number of clients assigned to a particular server when it is at the top of the preview window. In our example, an appropriate allocation of clients per scan weight may be assigned as follows: 21,1 client(10), 22,3 clients(10); 23,1 client (10); 24,2 clients(10); and 25,1(10) client. By using these scan weight values, the servers having the greatest capacity, i.e. 22 and 24, will be assigned to multiple clients before being removed from the preview window. Thus, the client requests are evenly distributed across all available servers 21-25 based on their capacity. In this example, four servers would still be available after the first ten client requests have been satisfied.

CLAIMS:

10. A computer implemented method for allocating a plurality of servers, each server having an available resource capacity, to a plurality of clients in a network for delivering a plurality of services to said clients, and for reducing the communication overhead on the network, the method comprising the steps of:

a) receiving a client request from one of the plurality of clients for one of said services in at least one broker;

b) selecting a subset of servers from the plurality of servers based on the available resource capacity of each of the plurality of servers; and

c) making a server suggestion, from the at least one broker, to one of the plurality of clients, the server suggestion identifying said subset of members to said one of the plurality of clients in response to said client request;

d) making a request for said one of said services from sad one of said clients to said suggested one of said servers in response to said one of said clients receiving the server suggestion of step c) from the at least one broker,;

e) operating the suggested one of said servers in accordance with its respective local policy to reject the request for said one of said services of step d) when the request exceeds the available resource capacity of the suggested one of said servers; and

f) operating the suggested one of said servers to accept the request for said one of said services of step d) when the request does not exceed the available resource capacity of the suggested one of said servers.